Title: Pavilion Basics

Author(s): Ferrell, Paul Steven
Sly, Nicholas Cameron
Lapid, Maria Francine Therese Ruiz

Intended for: This presentation will be used as part of training on the LANL owned (but open-sourced) Pavilion software.

Issued: 2019-12-19

Los Alamos NATIONAL LABORATORY EST.1943

PAVILION

HPC TEST HARNESS

Pavilion Tutorial

Paul Ferrell, Nick Sly, Francine Lapid

Pavilion Design Goals (Testing Types)

- Run system tests on HPC clusters
  - ‘Post DST’
  - Continuous
  - Acceptance
  - Software

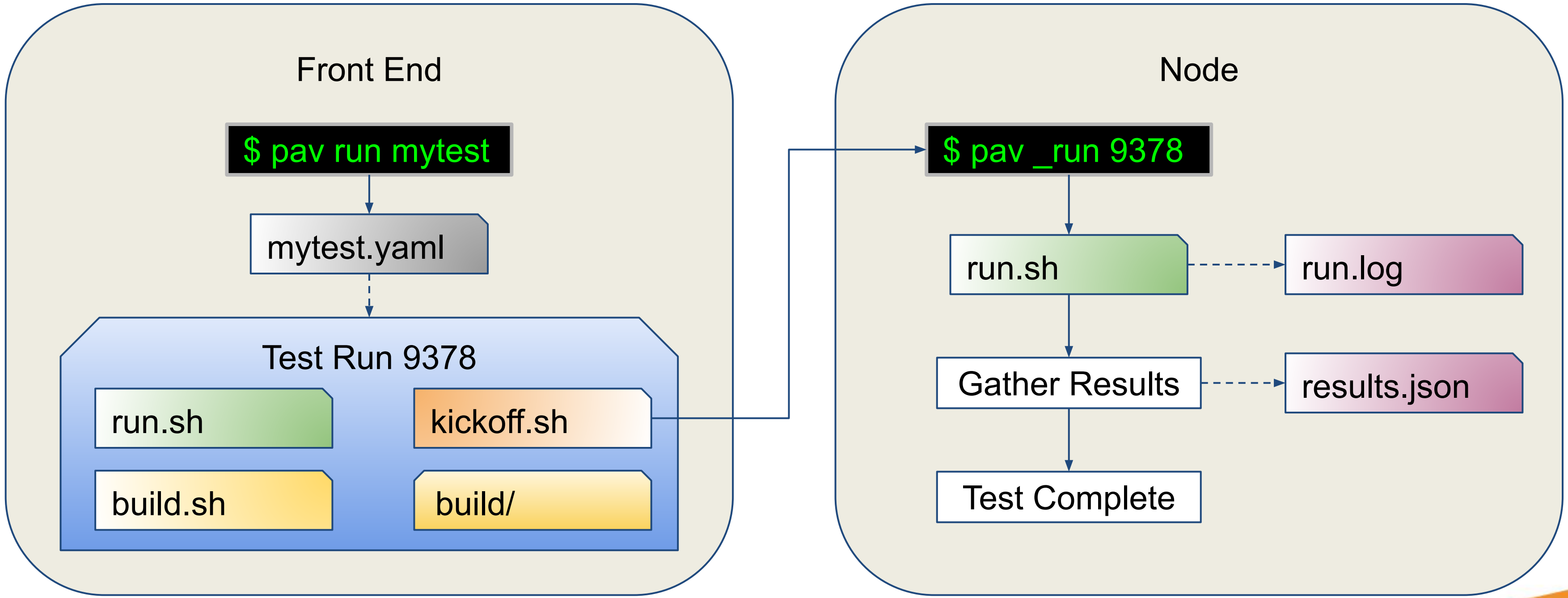
Pavilion Design Goals (Centralization)

- Centralize Common Problems/Solutions
  - Tests should work ‘everywhere’
  - Scheduling
  - Modules
  - Build/Run Environment
  - Result Parsing

Pavilion Design Goals (Stability)

- Introspection
- Easier Debugging
- Test Run Tracking
- Bug Tracking
  - Report any ‘exceptions’ encountered on https://github.com/hpc/pavilion2

Test Life Cycle



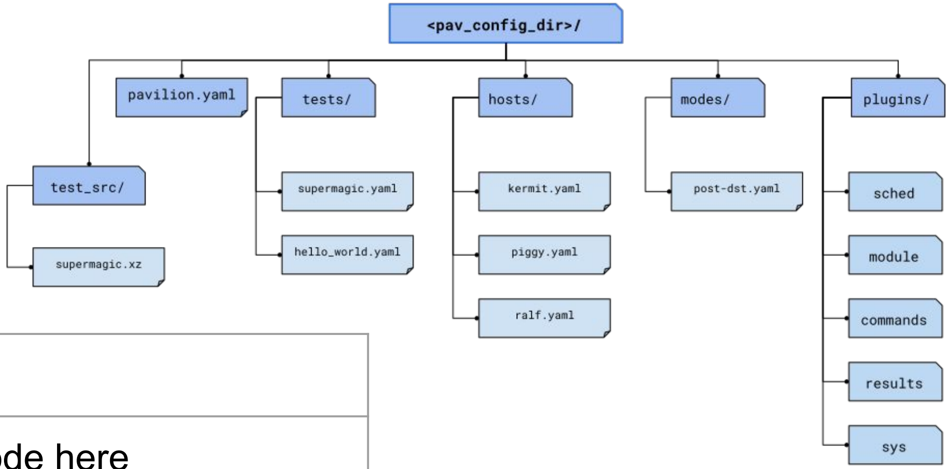
Los Alamos
NATIONAL LABORATORY
EST. 1943

PAVILION
HPC TEST HARNESS

Setup and Writing Your First Test

LANL HPC PRE-TEAM

Introduction



Directory	**Purpose**
`pavilion2/`	clone/install Pavilion source code here
`configs/`	configuration directory
`working_dir/`	where Pavilion writes the test runs, builds, and relevant logs

Install & Configure

Step 1	`git clone --recursive https://github.com/hpc/pavilion2.git`

Step 2	`export PAV_CONFIG_DIR=<pavilion config dir>`
	`export PATH=$PATH:<pavilion src dir>/bin`

Step 3	`<pavilion config dir>/pavilion.yaml`

pavilion.yaml

- working_dir
- shared_group
- result_log
- proxies

Write Test

supermagic

Step 1	Download zipfile of source (https://github.com/hpc/supermagic/zipball/master)
Step 2	Put the zip file in `<pavilion config dir>/test_src`
Step 3	Write the yaml config file (`<pavilion config dir>/tests`)

```
<pavilion config dir>/
    test_src/
        supermagic-master.zip
    tests/
        supermagic.yaml
```

Write Test (supermagic.yaml)

```
basic:


    summary: A basic supermagic run


    build:

        source_location: supermagic-master.zip


        cmds:
            - gcc -o supermagic supermagic.c
```

Test name.
Full name: supermagic.basic

Pavillion will auto-extract

multiple commands supported

Write Test (supermagic.yaml)

```
basic:
    ...

    run:

        cmds:
            - ‘{{sched.test_cmd}} ./supermagic’


    scheduler: slurm

    slurm:

        num_nodes: 2
        tasks_per_node: 2
```

Test name.
Full name: supermagic.basic

variable references need quotes

use Slurm to schedule job

Write Test (supermagic.yaml)

```
basic:
    ...

    results:

        regex:

            - key: num_tests

              regex: ‘num tests.* (\\d+)’


            - key: result

              regex: ‘<results> PASSED’
```

key is where result will be stored

Run Test

- `pav show tests`
  - shows available tests
- `pav run supermagic.basic`

… did it work?

Check logs

- `pav log [build | kickoff | run] test_id`

Fix Test (supermagic.yaml)

```
basic:
    ...

    build:

        modules: [gcc, openmpi/2.1.2]


        env:
            CC: mpicc


        source_location: supermagic-master.zip


        cmds:
            - ./autogen
            - ./configure
            - make
```

Pavilion will load these modules

Pavilion can set environment variables

Test Results

- `pav results`

Questions

Los Alamos
NATIONAL LABORATORY
EST. 1943

PAVILION
HPC TEST HARNESS

Advanced Usage

LANL HPC PRE-TEAM

Overview

- **Variables** - Dynamic test values
- **Permutations** - Iterative and scaled testing
- **Inheritance** - Copying and modifying tests
- **Hosts** - Host-specific settings
- **Modes** - Small, common settings

Variables - Overview

- **Variables** enable you to specify **values** for a given setting by individual...
  - test (test config)
  - host (host config & sys plugin)
  - scheduler (scheduler plugin)
  - pavilion installation (base pavilion installation)

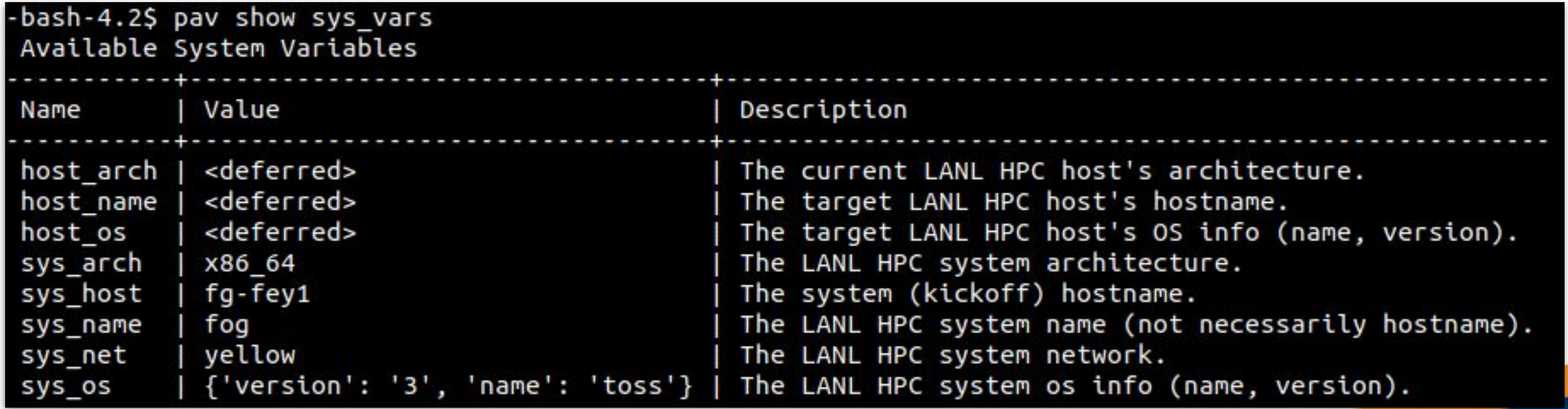
Variables - 4 Basic Types

Test Variables (var) - variables defined inside of a test suite, host config, or mode

```
basic:
    summary: “{{scratch1}}”
    …
    variables:
        scratch1: ‘/path/to/scratch1’
    …
```

Variables - 4 Basic Types

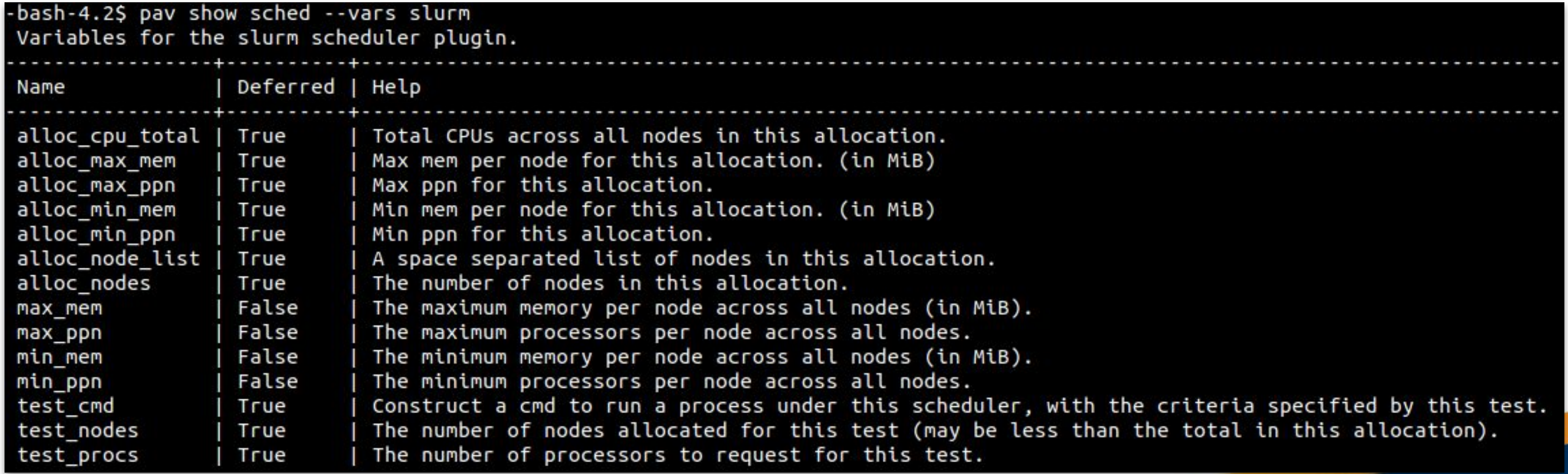
System Variables (sys) - Variables used to define system-specific values



```
-bash-4.2$ pav show sys_vars
 Available System Variables
-----------+-----------------------------------+------------------------------------------------
 Name      | Value                             | Description
-----------+-----------------------------------+------------------------------------------------
 host_arch | <deferred>                        | The current LANL HPC host's architecture.
 host_name | <deferred>                        | The target LANL HPC host's hostname.
 host_os   | <deferred>                        | The target LANL HPC host's OS info (name, version).
 sys_arch  | x86_64                            | The LANL HPC system architecture.
 sys_host  | fg-fey1                           | The system (kickoff) hostname.
 sys_name  | fog                               | The LANL HPC system name (not necessarily hostname).
 sys_net   | yellow                            | The LANL HPC system network.
 sys_os    | {'version': '3', 'name': 'toss'}  | The LANL HPC system os info (name, version).
```

Variables - 4 Basic Types

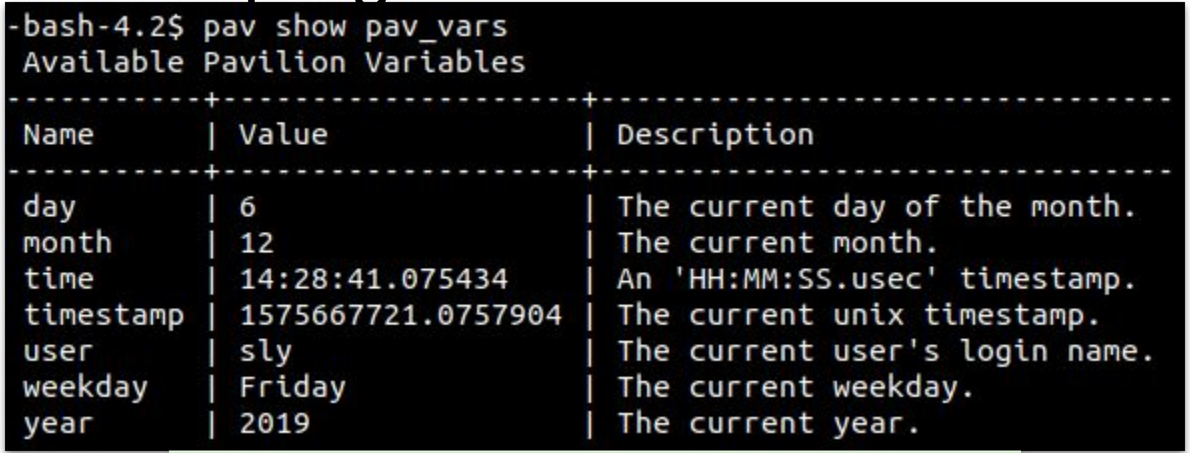
Scheduler Variables (sched) - Variables defining allocation-specific values.



```
-bash-4.2$ pav show sched --vars slurm
 Variables for the slurm scheduler plugin.
-----------------+----------+------------------------------------------------------------------------------------------------------
 Name            | Deferred | Help
-----------------+----------+------------------------------------------------------------------------------------------------------
 alloc_cpu_total | True     | Total CPUs across all nodes in this allocation.
 alloc_max_mem   | True     | Max mem per node for this allocation. (in MiB)
 alloc_max_ppn   | True     | Max ppn for this allocation.
 alloc_min_mem   | True     | Min mem per node for this allocation. (in MiB)
 alloc_min_ppn   | True     | Min ppn for this allocation.
 alloc_node_list | True     | A space separated list of nodes in this allocation.
 alloc_nodes     | True     | The number of nodes in this allocation.
 max_mem         | False    | The maximum memory per node across all nodes (in MiB).
 max_ppn         | False    | The maximum processors per node across all nodes.
 min_mem         | False    | The minimum memory per node across all nodes (in MiB).
 min_ppn         | False    | The minimum processors per node across all nodes.
 test_cmd        | True     | Construct a cmd to run a process under this scheduler, with the criteria specified by this test.
 test_nodes      | True     | The number of nodes allocated for this test (may be less than the total in this allocation).
 test_procs      | True     | The number of processors to request for this test.
```

Variables - 4 Basic Types

Pavilion Variables (pav) - Variables defined by the pavilion program itself.



```
-bash-4.2$ pav show pav_vars
 Available Pavilion Variables
-----------+---------------------+-------------------------------
 Name      | Value               | Description
-----------+---------------------+-------------------------------
 day       | 6                   | The current day of the month.
 month     | 12                  | The current month.
 time      | 14:28:41.075434     | An 'HH:MM:SS.usec' timestamp.
 timestamp | 1575667721.0757904  | The current unix timestamp.
 user      | sly                 | The current user's login name.
 weekday   | Friday              | The current weekday.
 year      | 2019                | The current year.
```

```
scratch1: '/path/to/scratch1/{{pav.user}}'
```

Variables - Usage

- Variables are always stored as **strings**
- Variables are referenced in quotes using **double curly braces**

```
subtitle: "{{scratch.name}}"
variables:
    scratch:
        - { name: scratch1, path: /path/to/scratch1/{{pav.user}} }
        - { name: scratch2, path: /path/to/scratch2/{{pav.user}} }
        - { name: scratch3, path: /path/to/scratch3/{{pav.user}} }
```

Variables - Usage

- Variables are a great way to generalize your tests.

```
- "{{sched.test_cmd}} ./supermagic -a -w {{scratch.path}}/"
```

- The test command above uses variables to <u>make a single command generic</u> to any of the ports as well as the scheduler.

Variables - Usage

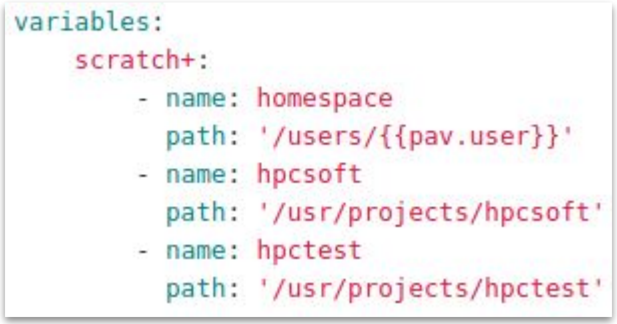
- When a variable contains a list of values, the entire list can be expanded by enclosing it in [~blah ~]

```
- “{{sched.test_cmd}} ./supermagic -a [~-w {{scratch.path}}/ ~]”
```

- This will expand to use the entire list
- The enclosed pattern is replicated for each element in the list
- A custom separator pattern can be placed between the final tilda (~) and square bracket (])

Variables - Usage

- Values can be added to a variable that may have been populated before

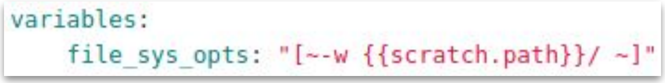


```yaml
variables:
    scratch+:
        - name: homespace
          path: '/users/{{pav.user}}'
        - name: hpcsoft
          path: '/usr/projects/hpcsoft'
        - name: hpctest
          path: '/usr/projects/hpctest'
```

- This will append the values to the existing variable
- The keys need to match the existing scheme
- These entries are <u>only added in the context of this file</u>

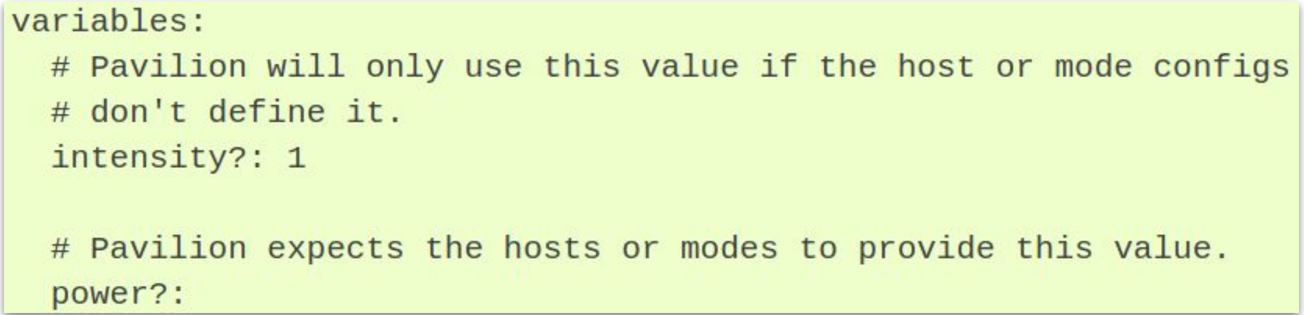
Variables - Usage

- Variables can be populated using other variables



```
variables:
    file_sys_opts: "[~-w {{scratch.path}}/ ~]"
```

- Variables can be populated if they haven’t already been set



```
variables:
  # Pavilion will only use this value if the host or mode configs
  # don't define it.
  intensity?: 1

  # Pavilion expects the hosts or modes to provide this value.
  power?:
```

Variables - Deferred

- Some variables can only be populated once an allocation has been granted
- These allow you to run your tests based on the allocation you've been **granted**, not what you **requested**
- E.g. - number of nodes, node list, PPN
- Deferred variables can only be used in certain parts of the test configs

Variables - Default Values

- When referencing a variable in a config, a default value can be provided if the variable hasn't been populated in two ways:

```
mytest:
    run:
        cmds:
            - "./mytest {{options|}} -m {{mode|simple}}"

complex_test:
    inherits_from: mytest

    variables:
        options: -a
        mode: complex
```

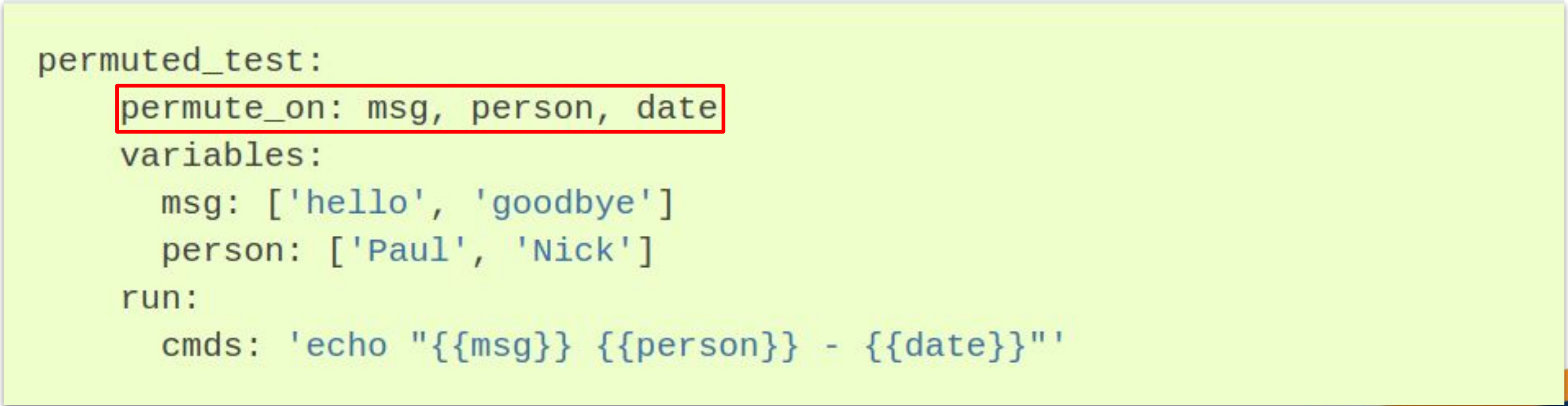
```
subtitle: "{{scratch.name}}"
variables:
    scratch?:
          - { name: scratch1, path: /path/to/scratch1/{{pav.user}} }
          - { name: scratch2, path: /path/to/scratch2/{{pav.user}} }
          - { name: scratch3, path: /path/to/scratch3/{{pav.user}} }
```

Permutations - Overview

- Enable having a single test config that generates several tests
- Leverages variables to permute over only the settings applicable at testing time
- Simplifies test configuration writing to cover all possibilities

Permutations - Usage

- Creates a new test for every combination of ‘msg’ (2), ‘person’ (2), and ‘date’ (1) = 4 tests



```
permuted_test:
    permute_on: msg, person, date
    variables:
      msg: ['hello', 'goodbye']
      person: ['Paul', 'Nick']
    run:
      cmds: 'echo "{{msg}} {{person}} - {{date}}"'
```

Permutations - Limitations

- You can not permute on sched variables
- You can not permute on deferred variables which won’t be resolved until after the permutations are generated
- There is no check for identical permutations, so you have to police yourself

Permutations - Complex Variables

- Permuting over complex variables can be useful

```
subtitle: "{{scratch.name}}"
permute_on: scratch
variables:
    scratch:
        - { name: scratch1, path: /path/to/scratch1/{{pav.user}} }
        - { name: scratch2, path: /path/to/scratch2/{{pav.user}} }
        - { name: scratch3, path: /path/to/scratch3/{{pav.user}} }
…
run:
    cmds:
        - '{{sched.test_cmd}} ./supermagic -a -w {{scratch.path}}'
```

Inheritance - Overview

Test inheritance allows for creating a **new test** by **copying** all of the configurations of **another test** and <u>only modifying the sections that differ</u>.

Inheritance - Rules

1. Copies all of the sections of another test in the same suite except for the ‘inherits_from’ key
2. Any section that is composed of a list will overwrite the entire list
3. A test can inherit from a test that inherited from another test
4. Inheritance is resolved before permutations

Inheritance - Example

```
base:
    permute_on: scratch
    variables:
        scratch:
            - { name: scratch1, path: /path/to/scratch1/{{pav.user}} }
            - { name: scratch2, path: /path/to/scratch2/{{pav.user}} }
            - { name: scratch3, path: /path/to/scratch3/{{pav.user}} }
    …
    run:
        cmds:
            - ‘{{sched.test_cmd}} -a -w {{scratch.path}}’
collective:
    inherits_from: base
    permute_on:
    variables:
        scratches: ‘[~-w {{scratch.path}}/ ~]’
    run:
        cmds:
            - ‘{{sched.test_cmd}} -a {{scratches}}’
```

Hosts - Overview

- Host configurations enable you to tell pavilion what **assumptions** it can make for a given **machine**
- Organized like a test config, but without the top-level test name

```
# HoneyBadger
variables:
    scratch:
        - name: scratch1
          path: "/path/to/scratch1/{{pav.user}}/"
        - name: scratch2
          path: "/path/to/scratch2/{{pav.user}}/"
    compilers: [gcc, intel, pgi]
    mpis: [openmpi, intel-mpi, mvapich2]

scheduler: slurm

slurm:
    num_nodes: 'all'
```

Modes - Overview

- Modes enable you to specify <u>small changes</u> that should be applied to <u>all tests</u> at runtime
- These are processed **after** the host and test configs

```yaml
slurm:
    reservation: PostDST
    qos: testers
```

Advanced Results

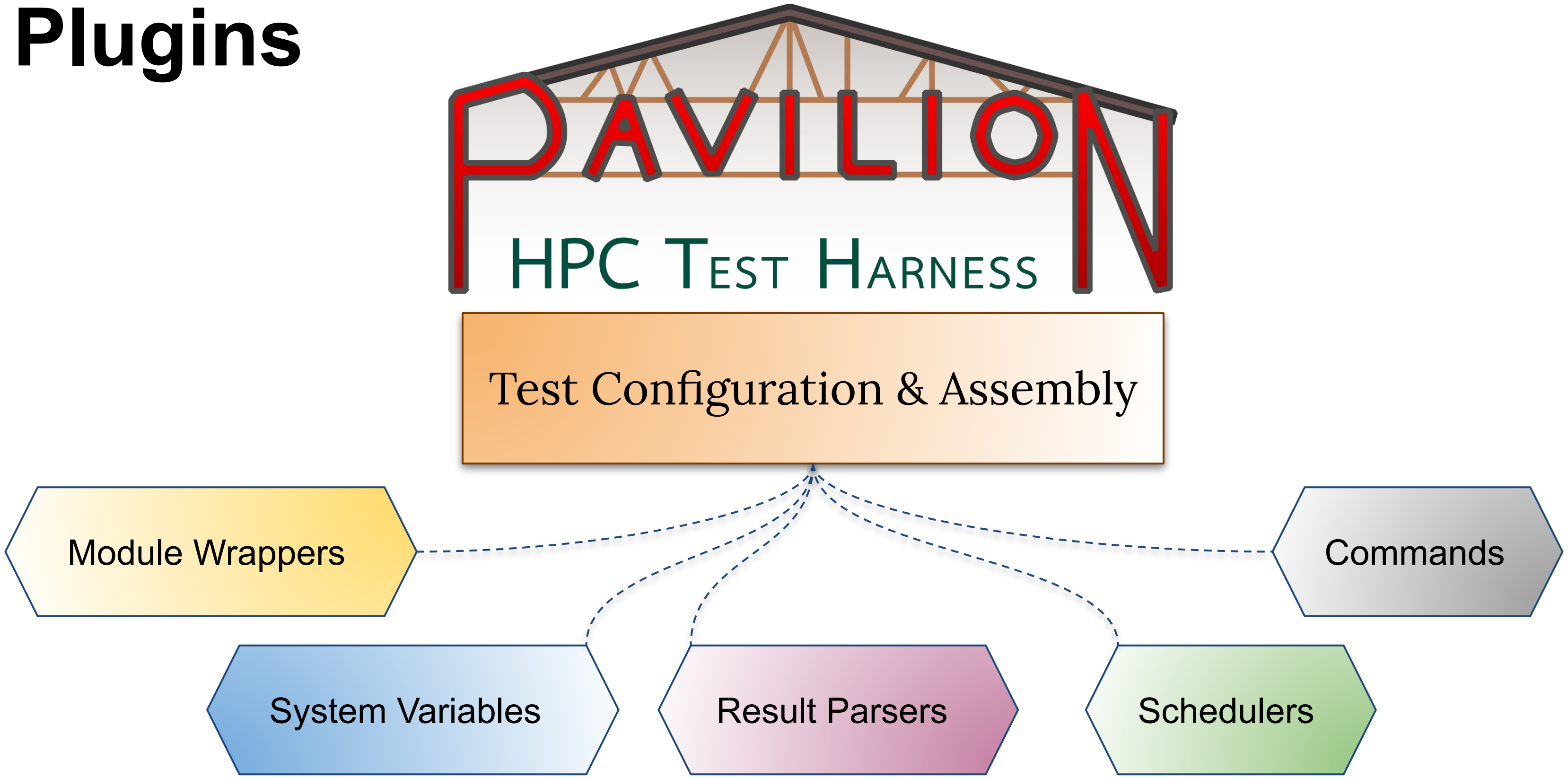
```
sometest:
     ...
     results:
          regex:
               - key: speed
                 regex: 'speed: (\d+)'
                 files: '*.out'
                 per_file: name
```

```
pav results -f

{
  'name': 'sometest'
  ...
  'per_name' :
     'n01':
        'speed': 55
     'n02':
        'speed': 63
}
```

Questions

Plugins


PAVILION
HPC Test Harness
Test Configuration & Assembly
Module Wrappers
Commands
System Variables
Result Parsers
Schedulers

Writing a System Plugin

```
$ pav show sys_vars
 Available System Variables
-----------+----------------------+----------------------------------
 Name      | Value                | Description
-----------+----------------------+----------------------------------
 host_arch | <deferred>           | The current host's architecture.
 host_name | <deferred>           | The target host's hostname.
 host_os   | <deferred>           | The target host's OS info (name, version).
 sys_arch  | x86_64               | The system architecture.
 sys_host  | durkula              | The system (kickoff) hostname.
 sys_name  | durkula              | The system name (not necessarily hostname).
 sys_os    | {'name': 'ubuntu',   | The system os info (name, version).
           |  'version': '18.04'} |
```

Writing a System Plugin

```
$ ls -l pavilion2/lib/pavilion/plugins/sys_vars/
total 60
-rw-r--r-- 1 bob bob  544 Jul 24 13:10 host_arch.py
-rw-r--r-- 1 bob bob  197 Jul 24 13:10 host_arch.yapsy-plugin
-rw-r--r-- 1 bob bob  541 Jul 24 13:10 host_name.py
-rw-r--r-- 1 bob bob  174 Jul 24 13:10 host_name.yapsy-plugin
-rw-r--r-- 1 bob bob  845 Jul 24 13:10 host_os.py
-rw-r--r-- 1 bob bob  181 Jul 24 13:10 host_os.yapsy-plugin
-rw-r--r-- 1 bob bob  550 Jul 24 13:10 sys_arch.py
-rw-r--r-- 1 bob bob  226 Jul 24 13:10 sys_arch.yapsy-plugin
-rw-r--r-- 1 bob bob  546 Jul 24 13:10 sys_host.py
...
```

~/.pavilion/plugins/sys_vars/sys_tz.py

```
from pavilion import system_variables
import subprocess

class DoesntMatter(system_variables.SystemPlugin):
    def __init__(self):
        super().__init__(
            name='sys_tz',
            description='The local timezone string.')
```

~/.pavilion/plugins/sys_vars/sys_tz.py

```
class DoesntMatter(system_variables.SystemPlugin):
    …

    def _get(self):
        date_str = subprocess.check_output(
            ['date', '+%Z'],
            stderr=subprocess.DEVNULL)

        return date_str
```

~/.pavilion/plugins/sys_vars/sys_tz.yapsy-plugin

```
[Core]
Name = Timezone
Module = sys_tz
```

~/.pavilion/plugins/sys_vars/sys_tz.py

```
class DoesntMatter(system_variables.SystemPlugin):
    …

    def _get(self):
        date_str = subprocess.check_output(
            ['date', '+%Z'],
            stderr=subprocess.DEVNULL)

        return date_str.decode(‘utf8’).strip()
```

Homework

- Override the ‘sys_name’ system variable with something useful.

Module Wrappers

FireChicken

```
module load gcc
module load openmpi

mpicc -o mytest mytest.c
```

ThunderCamel

```
module swap intel gcc
module swap intel-mpi openmpi

cc -G -o mytest mytest.c
```

~/.pavilion/plugins/module_wrappers/gcc.py

```
from pavilion import module_wrappers
from pavilion.module_actions import ModuleSwap

class Gcc(module_wrapper.ModuleWrapper):
    def __init__(self):
        super().__init__(
            name='gcc',
            description='Generic GCC wrapper',
            priority=self.PRIO_USER)
```

UNCLASSIFIED

~/.pavilion/plugins/module_wrappers/gcc.py

```
class Gcc(module_wrapper.ModuleWrapper):
    ...
    def load(var_man, requested_version=None):
        vers = self.get_version(requested_version)
        actions = []

        if sys_info.get('sys_name') == 'thunder_camel':
            actions.append(ModuleSwap('intel', '', 'gcc', vers))
            return actions, {}
        else:
            return super().load(var_man, requested_version)
```

~/.pavilion/plugins/module_wrappers/gcc.py

```
class OpenMPI(module_wrapper.ModuleWrapper):
    ...
    def load(var_man, requested_version=None):
        vers = self.get_version(requested_version)
        env = {}

        if sys_info.get(‘sys_name’) == ‘thunder_camel’:
            actions = [ModuleSwap(‘intel-mpi’, ‘’, ‘openmpi’, vers)]
            env[‘PAV_MPI_CC’] = ‘cc -G’
        else:
            actions, env = super().load(var_man, requested_version)
            env[‘PAV_MPI_CC’] = ‘mpicc’

        return actions, env
```

Result Parsers

run.log

```
1: PASSED
2: FAILED
3: SKIPPED
4: PASSED
5: PASSED
warning: no widgets found
6: FAILED
```

```
# passed
------------------- = percent_passed
# passed + # failed
```

~/.pavilion/plugins/results/percent_good.py

```
from pavilion import result_parsers
import yaml_config as yc
import re

class PercentGood(results_Parsers.ResultParser):
    def __init__(self):
        super().__init__(
            name='percent_match',
            description='Find the percent of items that match a regex.',
            priority=self.PRIO_USER)
```

~/.pavilion/plugins/results/percent_good.py

```
class PercentGood(results_Parsers.ResultParser):
    ...
    def get_config_items(self):
        conf_items = super().get_config_items()
        conf_items.extend([
            yc.StrElem(
                ‘good_re’,
                help_text=”Regex that matches ‘good’ items”),
            yc.StrElem(
                ‘bad_re’,
                help_text=”Regex that matches ‘bad’ items”)])
        return conf_items
```

~/.pavilion/plugins/results/percent_good.py

```
class PercentGood(results_Parsers.ResultParser):
    ...
    def _check_args(good_re=None, bad_re=None):
        try:
            re.compile(good_re)
        except re.error as err:
            raise result_parsers.ResultParserError(
                “Invalid regular expression: {}”.format(err))

        try:
            re.compile(bad_re)
        except re.error as err:
            raise result_parsers.ResultParserError(
                “Invalid regular expression: {}”.format(err))
```

~/.pavilion/plugins/results/percent_good.py

```
class PercentGood(results_Parsers.ResultParser):
    ...
    def __call__(self, test, file, good_re=None, bad_re=None):
        good_re = re.compile(good_re)
        bad_re = re.compile(bad_re)
        good = 0
        bad = 0
        for line in file.readlines():
            if good_re.search(line) is not None:
                good += 1
            elif bad_re.search(line) is not None:
                bad += 1
        return good/(good + bad)
```

Questions

Los Alamos
NATIONAL LABORATORY
EST. 1943

PAVILION
HPC Test Harness

Future Stuff

LANL HPC PRE-TEAM

Test Conditionals

```
mytest:
    only_if:
        sys_name: [honey_badger, fire_chicken]
```

Result Analyzer

```
mytest:
    results:
        parse:
            regex:
                -   key: speed
                    files: *.out
                    regex: ‘speed: (\d+)
                    per_file: by_name
        analyzer:
            outliers: ‘find_outliers(per_name.*.speed)’
            result: ‘len(outliers) < 2’
```

Series Files

```
prelim:
    tests:
        - license_check
        - mounts

main:
    depends_on: prelim
    tests:
        - supermagic
        - imb
```

Improved Concurrency

```
mytest:
    run:
        concur_cmd: ‘{{sched.test_single_cmd}} ./supermagic’
        concur_limit: 10
```

```
#!/bin/bash


...
for cmd in conc_cmds:
    ./srun -N 1 ./supermagic
```

More Improved Concurrency

- Shared Allocations

Future Questions